Apple Assembly Line

$1.80

Volume 6 -- Issue 11 August, 1986

About five minutes ago, about two hours before this issue goes to the printer, our UPS driver delivered one review copy of Programming the 65816, by David Eyes and Ron Lichty. IT'S HERE! And it looks excellent. I have time and space for only a few words here; we expect to have a complete review of this book and Fischer's next month.

Eyes and Lichty have given use over 600 pages of introduction, architecture, tutorial, application, and reference information, including a 60-page chapter listing and describing in detail the source code for a rudimentary 65816 Tracer/Debugger. This book has complete information on ALL the 65x family processors, from the original 6502 up through these latest 16-bit versions.

We expect delivery of our inventory within the next week or two and we'll start shipping your copies as soon as we get them. Remember, our price is $21 + shipping.

Minor Correction to Bob's New QUIT Code.............Erv Edge

Bob's new QUIT code for ProDOS is nice! There is one minor error at line 3390, which causes an extra character to be printed. Change "BNE .1" to "BPL .1" and it will work better. The extra character was only noticeable when the filename was 15 characters long, or had been RENAMEd to a shorter name than it previously had, because filenames are normally filled out with null characters.

You can also save one byte by changing line 4290 from "JMP MSG" to "BNE MSG". The BNE will always branch in this case, and is one byte shorter than the JMP.

Fast Integer Square Roots................Bob Sander-Cederlof

In the July 1986 issue of Dr. Dobb's Journal, Robert Pirko gave an algorithm and a program to calculate the square root of a 16-bit integer. (Letter, pages 10-12; program, pages 60-66.) His program was written in 8088 assembly language for the IBM PC. I decided to try writing the same algorithm in 6502 code.

As with most square root algorithms, Pirko's depends on the so-called Newton-Raphson iteration. In simple algebra, this means that a "pretty good" approximation to the square root of N can be made "much better" by the formula:

$$\text{much better} = \frac{1}{2}\left(\text{pretty good} + \frac{N}{\text{pretty good}} \right)$$

How many times you have to use the formula depends on how close your first guess is, and how much precision you need. By the definition of "integer square root", we are looking for the largest 8-bit integer whose square is less than or equal to the 16-bit argument N. Therefore, we need only 8-bit precision.

The formula refines the answer rapidly, roughly doubling the number of precise bits each time. So if we start with a close enough initial guess, once through the formula might be plenty.

What Pirko did was experiment to find a set of rules to get a "good enough" initial guess. Once through Newton-Raphson refined the guess to 8-bit precision, except that the truncation errors of integer arithmetic resulted in some roots being too large by 1. By squaring the calculated root and comparing the original number, Pirko could decrement those which needed it.

Pirko's program was pretty fast. He tested speed by taking the square roots of all integers from 0 to 65535, with the average time being 91 microseconds on his IBM PC (he didn't say which version, or what clock frequency). Any time I see a timed program like this, my benchmarking blood begins to bubble. However, the Apple is not as fast this time, because of the need for a division and a multiplication. The 8088 has opcodes for multiply and divide, while the 6502 does not.

Here are Pirko's rules for obtaining an initial guess:

Argument Range	Root or Initial Guess
0000-0001	root = N
0002-00FF	guess = N/16 + 3
0100-0FFF	guess = Nhigh*4 + 13
1000-7FFF	guess = Nhigh + 50
8000-D7FF	guess = Nhigh + 40
D800-FEFF	guess = 255
FF00-FFFF	root = 255

After playing with various versions of the program, I settled on a slightly different set of rules. The average time was

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners...................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM............. . .without source code $30, with source $50
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
                      Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)......................($49.95)  $45 *
Copy II Plus (Central Point Software)...................($39.95)  $35 *
Applesoft Toolbox Series (Roger Wagner Publishing).....each ($39.95)  $36 *
                                              all four ($159.80) $140 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks...........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL,
 saving you lots of typing and testing.
 The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $20 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers.............................  6 cents each

65802 Microprocessor (Western Design Center).................($95)  $50 *
quikLoader EPROM System (SCRG)..............................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *
Switch-a-Slot (SCRG)......................................($179.50) $170 *

"65816/65802 Assembly Language Programming", Fischer........($19.95)  $18 *
"Programming the 65816", Eyes...............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler......................($22.95)  $21 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95)   $9 *
"Microcomputer Graphics", Myers.............................($14.95)  $14 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95)  $18 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger........................($39.95)  $36 *

         * On these items add $2.00 for the first item and
           $.75 for each additional item for US shipping.
           Foreign customers inquire for postage needed.
     Texas residents please add 6 1/8 % sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
        ***                 (214) 324-2050                   ***
        *** Master Card, VISA, Discover and American Express ***
```

reduced a little by changing the third and fourth ranges:

```
0100-08FF   guess = Nhigh*4 + 13
0900-7FFF   guess = Nhigh + 50
```

It is not difficult to separate out the seven cases and calculate the initial guess. Lines 1130-1460 in the program below do all that work. Lines 1480-1530 perform the Newton-Raphson formula once. Lines 1540-1630 decrement the root if its square exceeds the original argument.

Since the 6502 does not have any multiply or divide opcodes, I coded these two operations as subroutines. My division subroutine divides the 16-bit value in ARGHI,ARGLO by the 8-bit value in GUESS, leaving the answer in QUOT and REM. My multiplication subroutine multiplies the 8-bit value in the A-register by the 8-bit value in ROOT to get a 16-bit product in PROD.

The program named MT (lines 3390-3850) does a complete test of the square root subroutine. MT calls on SQR for each possible argument from 0 through 65535 ($FFFF), then tests the value of ROOT by squaring it. If the square is larger than the argument, there was an error. If not, I increment the root and square it again. If this square is not larger than the argument, there was an error. If either type of error occurs, I display the argument and the root.

Once I was sure SQR was producing correct answers, I wrote a timing routine (lines 2110-2430) which runs through all values 0-65535 ten times. By changing line 1190 to "1190 SQR   RTS" I timed the overhead of the 655360 calls (13.2 seconds). When I changed line 1190 back, the additional time was 273.3 seconds. This is an average of 417 microseconds per root, over 4 times slower than the IBM program.

I wanted to see what the 6502 could do IF it had multiply and divide opcodes in hardware. Assuming each would take no more than 12 cycles, I changed lines 1740 and 1940 to RTS's. The resulting time averaged 74.3 microseconds per root. If only we had DIV and MUL, we could easily beat the PC in this benchmark.

We can come a little closer by playing a dirty trick. Since we know that the test program runs through all the arguments in sequence, we know that the squaring operation will usually be doing the same number over and over. For example, the root $FE comes out of the Newton-Raphson formula over 500 times! By giving the MUL routine a memory, we can save the root squared last time. Lines 1950-1970 are all we need to add. Simply remove the asterisks and re-assemble. With this trick, the average root took only 248 microseconds. We can also save about 24 microseconds per root by putting the MUL and DIV subroutines "in-line", eliminating the JSR and RTS (with the dirty trick, in-line code only saves an additional 12 microseconds).

Some time ago, Technical Education Research Centers (TERC) published information on the Hitachi 6203 multiply/divide chip and how to interface it to the Apple. We mentioned this in AAL a few years back. That chip allows you to code the division in

my program to run in 33 cycles, and the squaring in 30 cycles. The result would still be a little slower than the PC. By the way, all I know about the 6203 I got from TERC's newsletter. I have never been able to get any information whatsoever out of Hitachi about it, and I have never actually seen the chip. I sure would like to! (If you are interested in TERC, try calling Bob Tinker or his associates at (617) 547-0430, or write to TERC, 1696 Massachusetts Avenue, Cambridge, MA 02138.)

I am not finished yet. I spent another half day investigating a different technique for getting the integer square root. This is a binary adaptation of the mysterious method we learned and forgot in high school. Can you remember how we did it? I know at least a few of you can, because you are in high school right now. And others, because you are high school teachers. Anyway, it is too difficult to explain in these few lines. The same algorithm in binary is pretty straightforward. I present it in lines 2490-3020, without much comment. It works, but it takes an average of about 880 microseconds per root. The test routine in lines 3040-3350 takes the square root of evey value from 0-65535 using both SQR and NEWSQR, and compares the results. If they are different, it prints the data.

It is now nearly a month since I wrote all the preceding paragraphs in this article. Today the latest Dr. Dobbs Journal arrived, and a pair of letters referred to some old articles about integer square roots. I looked them up, and learned an easier way to apply the "high school" method. The resulting 6502 program won't fit in this issue, but is included on the Monthly and Quarterly Disks. This version takes an average of 737 cycles per root. It looks like a 65802 native mode version would be significantly faster.

Naturally, I expect at least one of you to come up with a significantly better and faster program than any of mine.

```
                  1000 *SAVE S.INTEGER.SQRT
                  
                  1020 *--------------------------------
00-               1030 ARGLO  .EQ 0
01-               1040 ARGHI  .EQ 1
02-               1050 GUESS  .EQ 2
03-               1060 QUOT   .EQ 3
04-               1070 REM    .EQ 4
05-               1080 ROOT   .EQ 5
06-               1090 PROD   .EQ 6,7
08-               1100 TRIPS  .EQ 8
09-               1110 PREVIOUS.ROOT .EQ 9
0A-               1120 BITHI  .EQ 10
0B-               1130 BITLO  .EQ 11
0C-               1140 SUBHI  .EQ 12
0D-               1150 SUBLO  .EQ 13
0E-               1160 WORKHI .EQ 14
0F-               1170 WORKLO .EQ 15
                  1180 *--------------------------------
                  1190 SQR
0800- A5 01       1200        LDA ARGHI
0802- 30 0E       1210        BMI .1         32768 OR MORE
0804- D0 18       1220        BNE .2         256...32767
0806- A5 00       1230        LDA ARGLO
0808- 4A          1240        LSR
0809- F0 3E       1250        BEQ .8         ...ARG= 0 OR 1
080B- 4A          1260        LSR
080C- 4A          1270        LSR
080D- 4A          1280        LSR
080E- 69 03       1290        ADC #3         (2...5 IS OKAY HERE
0810- D0 18       1300        BNE .4         ...ALWAYS
```

```
                 *---32768...65535----------------
0812- C9 FF      .1     CMP #255
0814- F0 30             BEQ .7        (A)=255=ROOT
0816- 69 28             ADC #40       (35...44 OKAY)
0818- 90 10             BCC .4
081A- A9 FF             LDA #255
081C- D0 0C             BNE .4        ...ALWAYS
                 *---256...32767------------------
081E- C9 09      .2     CMP #$09
0820- B0 06             BCS .3        $0900...$7FFF
0822- 0A                ASL           $0100...$08FF
0823- 0A                ASL
0824- 69 0D             ADC #13
0826- D0 02             BNE .4        ...ALWAYS
                 *---$0900...$7FFF----------------
0828- 69 31      .3     ADC #49       ADDS 50  (50...61 OKAY)
                 *---DO NEWTON ONCE---------------
082A- 85 02      .4     STA GUESS
082C- 20 4C 08          JSR DIV
082F- 18                CLC
0830- A5 03             LDA QUOT
0832- 65 02             ADC GUESS
0834- 6A                ROR
                 *---SQUARE THE RESULT------------
0835- 85 05             STA ROOT
0837- 20 67 08          JSR MUL
                 *---DECREMENT ROOT IF TOO BIG----
083A- A5 00             LDA ARGLO
083C- C5 06             CMP PROD
083E- A5 01             LDA ARGHI
0840- E5 07             SBC PROD+1
0842- A5 05             LDA ROOT
0844- E9 00             SBC #0        DECREMENT IF TOO BIG
                 *--------------------------------
0846- 85 05      .7     STA ROOT
0848- 60                RTS
                 *--------------------------------
0849- 2A         .8     ROL           RESTORE 0 OR 1
084A- 90 FA             BCC .7        ...ALWAYS
                 *--------------------------------
                 *   DIVIDE (ARGLO,ARGHI) BY (GUESS)
                 *      LEAVE ANSWER IN QUOT,REM
                 *--------------------------------
                 DIV
084C- A0 08             LDY #8
084E- A5 00             LDA ARGLO
0850- 85 03             STA QUOT
0852- A5 01             LDA ARGHI
0854- 06 03      .1     ASL QUOT
0856- 2A                ROL
0857- B0 04             BCS .15
0859- C5 02             CMP GUESS
085B- 90 04             BCC .2
085D- E5 02      .15    SBC GUESS
085F- E6 03             INC QUOT
0861- 88         .2     DEY
0862- D0 F0             BNE .1
0864- 85 04             STA REM
0866- 60                RTS
                 *--------------------------------
                 *   MULTIPLY (ROOT) BY (A-REGISTER)
                 *      PUT RESULT IN PROD,PROD+1
                 *--------------------------------
                 MUL
                 ***    CMP PREVIOUS.ROOT   "DIRTY TRICK"
                 ***    BEQ .7                ditto
                 ***    STA PREVIOUS.ROOT     ditto
0867- 85 02             STA GUESS
0869- A9 00             LDA #0
086B- A0 08             LDY #8
086D- 46 02      .5     LSR GUESS
086F- 90 03             BCC .6
0871- 18                CLC
0872- 65 05             ADC ROOT
0874- 6A         .6     ROR
0875- 66 06             ROR PROD
0877- 88                DEY
0878- D0 F3             BNE .5
```

```
087A- 85 07             STA PROD+1   SAVE HI-BYTE OF SQUARE
087C- 60          .7    RTS
                  *--------------------------------
                  T
087D- A9 0A             LDA #10
087F- 85 08             STA TRIPS
0881- A9 00             LDA #0
0883- 85 01             STA ARGHI
0885- 85 00             STA ARGLO
0887- 85 09             STA PREVIOUS.ROOT
                  *--------------------------------
0889- 20 00 08    .1    JSR    SQR
                    .DO 0
                        LDA ARGHI
                        JSR $FDDA
                        LDA ARGLO
                        JSR PRB
                        LDA ROOT
                        JSR PRB
                        LDA PROD+1
                        JSR $FDDA
                        LDA PROD
                        JSR PRB
                        JSR $FD8E
                    .FIN
                  *--------------------------------
088C- E6 00             INC ARGLO
088E- D0 F9             BNE .1
0890- EE F7 07          INC $7F7
0893- E6 01             INC ARGHI
0895- D0 F2             BNE .1
0897- EE F5 07          INC $7F5
089A- C6 08             DEC TRIPS
089C- D0 EB             BNE .1
089E- 60                RTS
                  *--------------------------------
089F- 20 DA FD    PRB   JSR $FDDA
08A2- A9 A0             LDA #" "
08A4- 4C ED FD          JMP $FDED
                  *--------------------------------
                  NEWSQR
08A7- A0 08             LDY #8       Loop 8 times for an 8-bit root
08A9- A5 01             LDA ARGHI
08AB- 85 0E             STA WORKHI   Save working copy of argument
08AD- A5 00             LDA ARGLO
08AF- 85 0F             STA WORKLO
08B1- A9 00             LDA #0       SUB0 = $4000
08B3- 85 0D             STA SUBLO    BIT0 = $4000
08B5- 85 0A             STA BITHI
08B7- A9 40             LDA #$40
08B9- 85 0C             STA SUBHI
08BB- 85 0A             STA BITHI
                  *--------------------------------
08BD- 38          .1    SEC          Trial subtraction
08BE- A5 0F             LDA WORKLO
08C0- E5 0D             SBC SUBLO
08C2- AA                TAX          Save lo-byte of difference
08C3- A5 0E             LDA WORKHI
08C5- E5 0C             SBC SUBHI
08C7- 90 04             BCC .2       ...WORKi < SUBi
08C9- 85 0E             STA WORKHI   Save new value for WORK
08CB- 86 0F             STX WORKLO
08CD- 08          .2    PHP          Save carry status (next ROOT bit)
08CE- 26 05             ROL ROOT     ROOT = ROOT*2 + CARRY
08D0- A5 0C             LDA SUBHI    SUB = (SUB .EOR. BIT)/2
08D2- 45 0A             EOR BITHI
08D4- 4A                LSR
08D5- 85 0C             STA SUBHI
08D7- A5 0D             LDA SUBLO
08D9- 45 0B             EOR BITLO
08DB- 6A                ROR
08DC- 85 0D             STA SUBLO
08DE- 28                PLP
08DF- 90 0C             BCC .3       ...WORK was less than SUB
08E1- A5 0C             LDA SUBHI    SUB = SUB .EOR. BIT
08E3- 45 0A             EOR BITHI
08E5- 85 0C             STA SUBHI
```

```
08E7- A5 0D                LDA SUBLO
08E9- 45 0B                EOR BITLO
08EB- 85 0D                STA SUBLO
08ED- 46 0A        .3      LSR BITHI     BIT = BIT/4
08EF- 66 0B                ROR BITLO
08F1- 46 0A                LSR BITHI
08F3- 66 0B                ROR BITLO
08F5- A5 0C                LDA SUBHI     SUB = SUB .EOR. BIT
08F7- 45 0A                EOR BITHI
08F9- 85 0C                STA SUBHI
08FB- A5 0D                LDA SUBLO
08FD- 45 0B                EOR BITLO
08FF- 85 0D                STA SUBLO
                   *--------------------------------
0901- 88                   DEY
0902- D0 B9                BNE .1
0904- 60                   RTS
                   *--------------------------------
                   TT
0905- A9 00                LDA #0
0907- 85 01                STA ARGHI
0909- 85 00                STA ARGLO
090B- 85 09                STA PREVIOUS.ROOT
                   *--------------------------------
090D- 20 A7 08     .1      JSR NEWSQR
0910- A5 05                LDA ROOT
0912- 48                   PHA
0913- 20 00 08             JSR SQR
0916- 68                   PLA
0917- C5 05                CMP ROOT
0919- F0 17                BEQ .11
091B- 48                   PHA
                     .DO 1
091C- A5 01                LDA ARGHI
091E- 20 DA FD             JSR $FDDA
0921- A5 00                LDA ARGLO
0923- 20 9F 08             JSR PRB
0926- A5 05                LDA ROOT
0928- 20 9F 08             JSR PRB
092B- 68                   PLA
092C- 20 9F 08             JSR PRB
092F- 20 8E FD             JSR $FD8E
                     .FIN
                   *--------------------------------
0932- E6 00        .11     INC ARGLO
0934- D0 D7                BNE .1
0936- EE F7 07             INC $7F7
0939- E6 01                INC ARGHI
093B- D0 D0                BNE .1
093D- 60                   RTS
                   *--------------------------------
                   *   COMPLETE TEST OF ALL POSSIBLE ARGUMENTS
                   *--------------------------------
                   MT
093E- A9 00                LDA #0
0940- 85 00                STA ARGLO
0942- A9 00                LDA #$00
0944- 85 01                STA ARGHI
                   *--------------------------------
0946- 20 00 08     .1      JSR SQR
0949- A5 05                LDA ROOT
094B- 20 67 08             JSR MUL
094E- A5 00                LDA ARGLO
0950- C5 06                CMP PROD
0952- A5 01                LDA ARGHI
0954- E5 07                SBC PROD+1
0956- 90 2B                BCC .9        ROOT TOO LARGE
0958- E6 05                INC ROOT
095A- F0 11                BEQ .2
095C- A5 05                LDA ROOT
095E- 20 67 08             JSR MUL
0961- C6 05                DEC ROOT
0963- A5 00                LDA ARGLO
0965- C5 06                CMP PROD
0967- A5 01                LDA ARGHI
0969- E5 07                SBC PROD+1
096B- B0 16                BCS .9        ROOT TOO SMALL
                   *--------------------------------
```

```
096D- E6 00     .2     INC ARGLO
096F- D0 D5            BNE .1
0971- E6 01            INC ARGHI
0973- A5 01            LDA ARGHI
0975- 4A               LSR
0976- 4A               LSR
0977- 4A               LSR
0978- 4A               LSR
0979- 09 B0            ORA #"0"
097B- 8D F7 07         STA $7F7
097E- A5 01            LDA ARGHI
0980- D0 C4            BNE .1
0982- 60               RTS
                *--------------------------------
0983- A5 01     .9     LDA ARGHI
0985- 20 DA FD         JSR $FDDA
0988- A5 00            LDA ARGLO
098A- 20 9F 08         JSR PRB
098D- A5 05            LDA ROOT
098F- 20 DA FD         JSR $FDDA
0992- 20 8E FD         JSR $FD8E
0995- 4C 6D 09         JMP .2
```

Updated Memory-vs.-File Maps for ProDOS...Bob Sander-Cederlof

I am not sure how it happened, but I seem to have botched up the table on page 20 of the November 1985 issue. As I now understand it, the relationship between the PRODOS file image (which loads at $2000) and the image of ProDOS after it is loaded is as follows (the lines marked with * are the changed lines):

	2000-287E	ProDOS Installer Code	
	287F-28FE	zeroes	
	28FF-293C	Installer for /RAM Driver	
	293D-29FF	zeroes	
	2A00-2BFF	Aux 200-3FF	/RAM/ Driver
*	2C00-2C99	FF00.FF99	/RAM/ Driver
	2C7F-2CFF		zeroes
*	2D00-4DFF	DE00-FEFF	MLI Kernel
	4E00-4EFF	BF00-BFFF	System Global Page
*	zeroes	D700-DDFF	
	4F00-4F7C	D742-D7BE	Thunderclock driver
	4F80-4FFF	FF80-FFFF	Interrupt Code
*	5000-56FF	D000-D6FF	Device Drivers
	5700-59FF	Alt D100-D3FF	QUIT Code

Looking at the same information from the viewpoint of the finished product, here is a map of ProDOS after it is loaded:

	4E00-4EFF	BF00-BFFF	System Global Page
*	5000-56FF	D000-D6FF	Device Drivers
*	zeroes	D700-DDFF	
	4F00-4F7C	D742-D7BE	Thunderclock driver
*	2D00-4DFF	DE00-FEFF	MLI Kernel
*	2C00-2C99	FF00-FF99	/RAM/ Driver
	4F80-4FFF	FF80-FFFF	Interrupt Code
	5700-59FF	Alt D100-D3FF	QUIT Code

DUMP Command for DOS 3.3............................Bill Morgan

Back in March, 1984 Bob S-C wrote up a modification to the DOS VERIFY command processor to make that command do a hex dump of a file. This can be a very useful tool, giving us the ability to see exactly what's stored in any file of any type, so of course I wanted to include it in the DOS I've been building for the UniDisk 3.5. In his version Bob provided a 40-column hex-only display and invited the reader to expand it to 80 columns with ASCII as well. That's what I've done here, as well as adding pause and abort features. I didn't get as far as his suggestion to allow paging through the file either backwards or forwards, so there's still something left for you tinkerers.

As written here DUMP runs in 158 bytes of page 3. I got involved in making it better, rather than making it smaller, so it still won't fit inside of DOS yet. Maybe next time around.

There is a side effect of modifying VERIFY. Both SAVE and BSAVE call VERIFY after they're done, to make sure all went well. The first time I got this patch working and then SAVEd the file I got a nice dump all over the screen at the end of the SAVE. That's why I ended up having the DUMP command call a little routine that patches into the VERIFY code, calls it, and then de-patches itself. This means that if you RESET out of a DUMP, or if you get an I/O ERROR, the DUMP patch will still be in place and you'll probably want to disconnect it. You can do that either by doing a successful DUMP command, or by calling the disconnect routine directly ($329G or CALL 809).

So, on to the code. By the way, I use a couple of 65C02 opcodes in this program, but the equivalent 6502 instructions are shown {in curly brackets} in the comment field.

INSTALL points the command vector to my routine and changes the command name from VERIFY to DUMP. Normally when you change a DOS command name the new name must have the same number of letters as the old one, but since VERIFY is the last name in the list we can truncate it and fill the ending bytes with zeroes.

PATCH is called when we type the DUMP command. Here we patch into the VERIFY processor to call DUMP in place of the call to get the next sector of the file. It then calls VERIFY as usual and then puts it back. This way we can get the sector and display it when it's our command, but leave the normal VERIFY operation undisturbed so SAVE and BSAVE can function normally.

DUMP first does the GET.SECTOR call we patched in over, exiting when we hit the end of the file. The next step is to prepare the display so we print a blank line and display the relative sector number in the file. Note that this is not a disk track or sector number; this is the position in the track/sector list. We then set the Y-register to the beginning of the sector buffer and branch into the code to display one line.

The line display routines starting at line 1600 normally begin by printing four spaces to allow for the sector number printed on the first line. After that we print the offset into the

sector from the Y-register and a separating dash. Notice that the PRINT.BYTE and PRINT.DASH routines also output a trailing space. Now it's time to get the next 16 bytes from the file, stash them in a buffer for later ASCII printing, and display them in hexadecimal. (The stash buffer is inside the secondary filename buffer, which is used only during a RENAME command.) The odd code in lines 1730-1780 has the effect of printing an extra space after every four bytes, to separate the display into columns and improve readability.

The last steps in displaying a line are to recover the 16 bytes from the buffer, make them printable and replace any control characters with underlines, and send them on out. (Most programs seem to replace control characters with periods, but I chose the underline so I could more easily see the real periods in the file. Underlines are much rarer in typical text.) After that we check the keyboard for a pause or abort and check to see if we've finished the sector yet. If necessary we branch back and do the next line.

PAUSE is lifted straight from the SHOW command article in AAL July, 1982. As a matter of fact, since I always install SHOW I just call PAUSE at $AE8E rather than including it in DUMP. This is a very useful routine to keep around.

```
                        1000 *SAVE S.DUMP
                        1010        .OP 65C02
                        1020 *--------------------------------
                        1030 *
                        1040 *      Patch DOS to change
                        1050 *      VERIFY into DUMP
                        1060 *
                        1070 *--------------------------------
42-                     1080 POINTER     .EQ $42
9D54-                   1090 VECTOR      .EQ $9D54
A27D-                   1100 VERIFY      .EQ $A27D
A902-                   1110 VFY.NAME    .EQ $A902
AAA0-                   1120 BUFFER      .EQ $AAA0
AD1C-                   1130 READ.CALL   .EQ $AD1C
B0B6-                   1140 GET.SECTOR  .EQ $B0B6
B5E4-                   1150 SECTOR      .EQ $B5E4
C000-                   1160 KEYBOARD    .EQ $C000
C010-                   1170 STROBE      .EQ $C010
F941-                   1180 MON.PRNTAX  .EQ $F941
F94A-                   1190 MON.PRBL2   .EQ $F94A
FD8E-                   1200 MON.CROUT   .EQ $FD8E
FDDA-                   1210 MON.PRBYTE  .EQ $FDDA
FDED-                   1220 MON.COUT    .EQ $FDED
                        1230 *--------------------------------
                        1240        .OR $300
                        1250 *      .TF B.DUMP
                        1260 *--------------------------------
0300- A9 03             1270 INSTALL LDA /PATCH-1      point DOS vector table
0302- 8D 55 9D          1280        STA VECTOR+1      to my patch
0305- A9 1B             1290        LDA #PATCH-1
0307- 8D 54 9D          1300        STA VECTOR
030A- A2 05             1310        LDX #5
030C- BD 16 03          1320 .1     LDA COMMAND,X     change VERIFY command
030F- 9D 02 A9          1330        STA VFY.NAME,X    name to DUMP
0312- CA                1340        DEX
0313- 10 F7             1350        BPL .1
0315- 60                1360        RTS
0316- 44 55 4D
0319- D0                1370 COMMAND .AT /DUMP/
031A- 00 00             1380        .HS 0000
                        1390 *--------------------------------
031C- A9 34             1400 PATCH  LDA #DUMP         hook into VERIFY command
031E- 8D 1C AD          1410        STA READ.CALL
0321- A9 03             1420        LDA /DUMP
0323- 8D 1D AD          1430        STA READ.CALL+1
0326- 20 7D A2          1440        JSR VERIFY        call it
```

```
0329- A9 B6     1450        LDA #GET.SECTOR    restore normal VERIFY
032B- 8D 1C AD  1460        STA READ.CALL
032E- A9 B0     1470        LDA /GET.SECTOR
0330- 8D 1D AD  1480        STA READ.CALL+1
0333- 60        1490        RTS
                1500 *-------------------------------
0334- 20 B6 B0  1510 DUMP   JSR GET.SECTOR     read next sector
0337- B0 59     1520        BCS .7             end of file
0339- 20 8E FD  1530        JSR MON.CROUT      start sector with <CR>
033C- AD E5 B5  1540        LDA SECTOR+1
033F- AE E4 B5  1550        LDX SECTOR
0342- 20 41 F9  1560        JSR MON.PRNTAX     display sector position
0345- A0 00     1570        LDY #0             start at beginning of sector
0347- F0 05     1580        BEQ .2          ...always
                1590
0349- A2 04     1600 .1     LDX #4             print 4 blanks
034B- 20 4A F9  1610        JSR MON.PRBL2      so columns look neater
034E- 98        1620 .2     TYA
034F- 20 9B 03  1630        JSR PRINT.BYTE     print byte count
0352- 20 94 03  1640        JSR PRINT.DASH     separator
0355- A2 0F     1650        LDX #15            16 bytes per line
0357- B1 42     1660 .3     LDA (POINTER),Y    get byte from file
0359- 9D A0 AA  1670        STA BUFFER,X       stash it
035C- 20 9B 03  1680        JSR PRINT.BYTE     print as hex value
035F- C8        1690        INY                next byte
0360- CA        1700        DEX
0361- 30 0B     1710        BMI .4             done with this line?
                1720
0363- 8A        1730        TXA                check X
0364- 1A        1740        INC                {EOR #%00000011}
0365- 29 03     1750        AND #%00000011
0367- D0 EE     1760        BNE .3             every fourth byte
0369- 20 9E 03  1770        JSR PRINT.SPACE     skip a space
036C- 80 E9     1780        BRA .3             {JMP .3}
                1790
036E- 20 94 03  1800 .4     JSR PRINT.DASH     separator
0371- A2 0F     1810        LDX #15            16 bytes
0373- BD A0 AA  1820 .5     LDA BUFFER,X       get stashed value
0376- 09 80     1830        ORA #%10000000     hi-bit on
0378- C9 A0     1840        CMP #" "
037A- B0 02     1850        BCS .6             filter out controls
037C- A9 DF     1860        LDA #"_"           substitute "_"
037E- 20 ED FD  1870 .6     JSR MON.COUT       print as ASCII
0381- CA        1880        DEX
0382- 10 EF     1890        BPL .5             back for more
                1900
0384- 20 8E FD  1910        JSR MON.CROUT      next line
0387- 20 A3 03  1920        JSR PAUSE          check for pause or
038A- F0 06     1930        BEQ .7              abort
038C- C0 00     1940        CPY #0             done with sector?
038E- D0 B9     1950        BNE .1
0390- 18        1960        CLC                normal exit
0391- 60        1970        RTS
0392- 38        1980 .7     SEC                EOF or abort exit
0393- 60        1990        RTS
                2000 *-------------------------------
                2010 PRINT.DASH
0394- A9 AD     2020        LDA #"-"
0396- 20 ED FD  2030        JSR MON.COUT
0399- 80 03     2040        BRA PRINT.SPACE    {JMP PRINT.SPACE}
                2050 PRINT.BYTE
039B- 20 DA FD  2060        JSR MON.PRBYTE
                2070 PRINT.SPACE
039E- A9 A0     2080        LDA #" "
03A0- 4C ED FD  2090        JMP MON.COUT
                2100 *-------------------------------
                2110 *      RETURN .EQ. IF ABORT
                2120 *             .NE. IF CONTINUE
                2130 *-------------------------------
03A3- AD 00 C0  2140 PAUSE  LDA KEYBOARD       any key pressed?
03A6- 10 11     2150        BPL .2             no, continue
03A8- 8D 10 C0  2160        STA STROBE         yes, clear strobe
03AB- C9 8D     2170        CMP #$8D           abort?
03AD- F0 0A     2180        BEQ .2             yes, return .EQ. status
03AF- AD 00 C0  2190 .1     LDA KEYBOARD       no, pause 'til keypress
03B2- 10 FB     2200        BPL .1             none pressed yet
03B4- 8D 10 C0  2210        STA STROBE         clear strobe
03B7- C9 8D     2220        CMP #$8D           abort?
03B9- 60        2230 .2     RTS                .EQ. if abort
```

Compatibility with the Laser-128.........Bob Sander-Cederlof

We borrowed a Laser-128 (popular clone of the Apple //c) the other day. It had been rumored that our software would not run on it, in spite of Central Point Software's sanguine claims. Sure enough, the S-C Macro Assembler would not operate, under either DOS or ProDOS. They boot and load, but no more.

A little investigation revealed what we expected: our software uses at least a half-dozen entry points into the Apple monitor which are not supported in the Laser-128 monitor. Most of them have to do with our "$" command, which lets you perform monitor commands without leaving the S-C environment. These patches will disable the "$" command and repair the "MEM" command. The addresses shown are for our current release disks.

```
DOS 3.3 $1000 version    1AE6:4C B3 1B 20 40 F9 A9 AD 4C ED FD
                         124A:E9 1A  (was 99 FD)
                         125D:E9 1A  (was 99 FD)

DOS 3.3 $D000 version    DAE6:4C B3 DB 20 40 F9 A9 AD 4C ED FD
                         D24A:39 DA  (was 99 FD)
                         D25D:E9 DA  (was 99 FD)

ProDOS version           8B45:4C 24 8C 20 40 F9 A9 AD 4C ED FD
                         8450:48 8B  (was 99 FD)
                         8463:48 8B  (was 99 FD)
```

Make a backup copy of the disk, and then boot the backup copy. When the assembler version you choose has loaded, type the letter X and the RETURN key. This should BRK out of the assembler into the Laser-128 monitor. Make the patches as shown above, and then type "3D0G" or control-RESET to get back into the assembler. It should be working correctly now. If you are fixing the DOS 3.3 version, you can now BSAVE the patched code on the file you originally loaded.

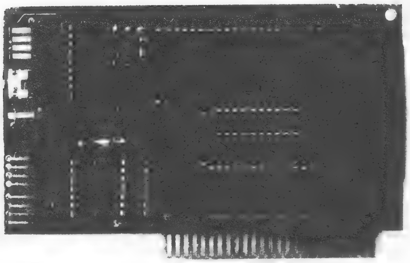
If you are fixing the ProDOS version, you now should BLOAD the type SYS file called SCASM.SYSTEM. The same patches you just made to the assembler should now be applied to the image of the SYS file, and then BSAVE the image on the disk:

```
:BLOAD SCASM.SYSTEM,TSYS,A$2000
:MNTR
*2D45:4C 24 8C 20 40 F9 A9 AD 4C ED FD
*2650:48 8B
*2663:48 8B
*3D0G
:BSAVE SCASM.SYSTEM,TSYS,A$2000,L17920
```

One incompatibility remains for which we never found the cause: the esc-L shorthand command, to turn a CATALOG line into a LOAD command, does not work in 80-column mode. It does work just fine in 40-column mode. If any of you try these patches and find other problems, we would like to hear about them.

One more item: we found the Laser-128 monitor incorrectly disassembles the PLX command as PHX.

The first clock card to add time and date to AppleWorks'™ screens and files.



Timemaster H.O. So many features its become the world's best selling clock card.

It's the smart way to put the time and date on your Apple IIe* or II+*. Because only the Timemaster H.O. packs *ALL* the important features of all the competition *combined*, including leap year, year (not just in PRO DOS), month, date, day of week, hours, minutes, seconds and milli seconds. It's totally PRO-DOS, DOS 3.3, PASCAL and CP/M compatible. And, of course, it works better than any other clock with AppleWorks.

With the Timemaster H.O., you're assured of reliable, trouble-free operation along with simple, one-command time setting, accuracy through leap years, and a firmware maintained interrupt setting.

If you're using or writing software for other clock cards, you're still covered. Because the H.O. will *automatically* emulate them. And only the Timemaster H.O. adds 15 new commands to BASIC. The H.O. even comes complete with two disks full of sample programs, including a computerized appointment book, a DOS dating program, interrupt programs, and over 30 programs that others charge you extra for — or don't even offer. As a low cost option, you can add true BSR remote control to the H.O., giving you remote control of up to 16 lights and appliances in your home or office.

• Time in hours, minutes, seconds and milliseconds, date with year, month, day of week and leap year • 24-hour military format or 12-hour AM/PM format • Eight software controlled interrupts so you can run two programs at the same time (many examples included) • Can be set to the second • Allows AppleWorks to time and date stamp all data automatically • Displays time and date on AppleWorks screens and files • Five-year warranty

Timemaster H.O **$129.00**
BSR option (may be added later) **$ 39.00**

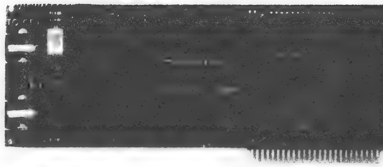
Call to order today, 9 a.m. to 11 p.m. seven days, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 outside U.S.A.

ÆE Applied Engineering
The Apple enhancement experts.

P.O. Box 798, Carrollton, TX 75006 (214) 241-6060

Viewmaster 80

Now run AppleWorks™ on your II+ *and* keep full 80 column compatibility!



	BUILT-IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	80 COLUMN HOME	7 x 9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS	
VIEWMASTER 80	✓	✓	✓	✓	✓	✓	✓	✓	
SUPRTERM		✓					✓	✓	
WIZARD 80					✓		✓	✓	
VISION 80	✓	✓			✓				
OMNIVISION		✓					✓	✓	
VIEWMAX 80	✓	✓			✓			✓	
SMARTERM	✓	✓				✓	✓		
VIDEOTERM		✓	✓		✓	✓		✓	

One look at the chart will give you some of the reasons there's only one smart choice in 80 column cards for your Apple. But the real secret to Viewmaster 80's success is something even better: Total compatibility.

Each Viewmaster 80 includes our powerful AppleWorks expand software, allowing AppleWorks to run on the II+ with only 64K (or more) memory. (We recommend the RamFactor memory card, but any compatible Apple memory card will work.) The software provides our full range of AppleWorks enhancements, including expanded records, word processor, multi-disk saving, time and date display on screen with any PRO-DOS clock, and more!

The Viewmaster 80 works with all 80 column applications, including DOS 3.3, PRO-DOS, CP/M, Pascal, WordStar, Format II, Applewriter II, dBase II, Visicalc, Multiplan, and ***hundreds*** of others — including AppleWorks.

Here are just a few of the powerful features the Viewmaster 80 delivers for a great price (**$**139):
• 80 Characters by 24 lines • Fully compatible with all Apple languages and software • Highest compatibility with existing 80 column software • Very low power consumption • High speed (18 MHZ) scroll rate • Upper and lower case characters and true descenders, both inverse and normal; all on-screen editing functions are supported • User-definable cursor shape • Compatible with Apple II, II+ and IIe • Five-year warranty

Call today to order or for more information, 9 a.m. to 11 p.m. seven days, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add **$**10.00 outside U.S.A.

ÆE Applied Engineering
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

Convert Lo-Res Pictures to Double Hi-Res.....David C. Johnson
(with Info about Two Secret RGB Modes)

In the January '86 AAL, Bob presented a routine to convert lo-res to regular hi-res. He pointed out that "it is not possible to exactly reproduce the lo-res colors on the hi-res screen (unless I used //e or //c double hi-res)." Bob's routine used patterns to "come close to the same color." He also wrote that he didn't have a color monitor and didn't know how close he got. I have an RGB color monitor, and patterns result in most unsatisfactory conversion. This article offers three satisfactory conversion routines that, of course, use double hi-res.

Regular color double hi-res, which will work on a color TV, a composite color monitor, or an RGB monitor (mode 2 or 3), has a resolution of 140 by 192. Lo-res has a resolution of 40 by 48. Since 140 is not evenly divisible by 40, regular color double hi-res is not capable of displaying a satisfactory representation of a lo-res picture. You can reproduce all sixteen colors, but you can't get satisfactory resolution of 40 lo-res columns. An RGB monitor, driven by an AppleColor (or compatible) card, can display a PERFECT double hi-res version of a lo-res picture two different ways. Two of my routines perform these color conversions and work ONLY for RGB. Without RGB they look lousy. The third routine, included for completeness, performs a monochrome conversion and will work on an RGB monitor (mode 1) or a monochrome monitor.

I'm writing this article so I may present examples of how the RGB-only video modes work. The two color routines use modes that are not described in the AppleColor card manual. The AppleColor card comes with demo disks that show foreground/background hi-res; a Video-7 demo disk I have shows "160 Mode". Except for these two simple demos and a collection of programs which I have written, I know of no programs, commercial or otherwise, using these modes. By the way, while I do have a //c, I don't have an RGB interface for it, so I don't know if Video-7's (or anyone else's) //c RGB Adapter supports these modes or is compatible with the AppleColor card. I would be interested in hearing from any of you on these subjects at: 41 Putnam Park Road, West Redding, CT 06896.

F/B hi-res works much like monochrome regular hi-res. Main memory bytes define the screen at seven pixels each (bit 7 is ignored - no half shifting), but instead of each 1 bit producing an "on" pixel and each 0 bit an "off" pixel, each 1 bit produces a "foreground" pixel and each 0 bit a "background" pixel. Auxiliary memory bytes define the F/B colors of the seven pixels at the same address in main memory. The high nybbles contain the foreground colors and the low nybbles the background colors. For example, aux $2000:12 and main $2000:55 would produce magenta, dark blue, magenta, dark blue, magenta, dark blue, and magenta beginning at the top left. In effect you get 280x192 resolution with sixteen colors! Of course, since F/B colors are defined for each group of seven pixels, not each pixel, you can only get two colors per seven pixels.

With "160 Mode", the "given" of using only seven bits of every

byte for pixel information is gone. All eight bits of both the auxiliary and main memory pages are used. It seems (to me) to be more closely related to lo-res than hi-res, only "sideways" and with sixteen times the resolution. Low nybbles define the even columns and high nybbles the odd columns, while in lo-res the correspondence is between nybbles and rows. Auxiliary memory pairs precede main memory pairs. For example, aux $2000:21 and main $2000:43 would produce: magenta, dark blue, purple, and dark green beginning at the top left. Effective resolution is 160x192 with sixteen colors and no restrictions!

Both the RGB modes offer superior pictures compared to regular hi-res and (regular) double hi-res. They also offer other advantages important to programmers. We can manipulate F/B hi-res images pixel data separately from the color data. "160 Mode" images can be drawn in 80 different horizontal positions without any shifting, and the other 80 are only a 4 bit shift (or look up) away using whole bytes. Best of all, the bits that define a colored pixel don't ever fall in different bytes!

The three routines may be tested by entering at labels "a", "b" and "c". These are extensions the demo "T" in Bob's program. I have also kept his "PLOT" and "PAUSE.FOR.ANY.KEY" routines. I've rearranged Bob's code to handle some //e (and //c) stuff, the double hi-res video modes, etc. You should refer back to his article, and compare our code. One thing that I didn't carry forward from Bob's code is his stepping through the screen memory using Cartesian coordinates. My routines crank along sequentially. The video mode switching is all handled by these test routines.

F/B mode is active any time the 80 column switch is off and annunciator 3 is off. The other double hi-res RGB modes are activated by clocking two data bits and a "1" bit into a shift register on the AppleColor card. The value of the data bits determine which mode is activated. The data bits and the "1" bit are the setting of the 80 column switch at the time the shift register is clocked. The shift register is clocked by lowering the annunciator 3 switch.

ConvertLoResToFBHiRes first sets the main memory pixel buffer to a pattern of 24 black horizontal regions (0, "off") with 24 white bars (1, "on") interspaced. The "off" (excuse me, background) areas correspond to the even lo-res rows, and the "on" (foreground) to the odd rows. This works out to filling main $2XXX w/$00 and main $3XXX w/$7F. By using this pattern, I avoided having to manipulate the lo-res data before storing it in auxiliary memory. To understand this you must remember that even lo-res rows are stored in low nybbles and odd lo-res rows in high nybbles. After setting up main memory, the lo-res data is expanded 8:1 into the auxiliary memory color buffer. For example: $400 gets copied to aux $2000, $2400, $2800, $2C00, $3000, $3400, $3800, and $3C00. If this is viewed on a non-RGB monitor, all you'll see is 24 bars.

ConvertLoResToDoubleHiRes160 expands the lo-res data nybbles 16:1. Each lo-res byte is split into low and high nybbles. Each nybble is in turn duplicated in the other half of 8 bits

and then copied into the main and auxiliary memory locations that correspond to the lo-res pixel the nybble used to be. For example: the low nybble of $400 gets copied to both nybbles of aux $2000, $2400, $2800, $2C00, and main $2000, $2400, $2800, $2C00; the high nybble of $400 gets copied to both nybbles of aux $3000, $3400, $3800, $3C00, and main $3000, $3400, $3800, $3C00. If you look at this on a monochrome monitor you'll see only 7/8ths of the data; it looks strange. Viewed on a non-RGB color screen, you'll get a mess of wrong colors.

The last routine, ConvertLoResToDoubleHiRes560, draws double hi-res monochrome patterns (NOT color patterns). I carefully studied the patterns lo-res makes when viewed on a monochrome monitor, writing down the constituent binary numbers. From that, I typed in the table "Monos", extending the patterns to include both even and odd offsets, and dividing them into auxiliary and main bytes. Once the table was defined, the routine just fell together ("automatically")! It uses each nybble and its Y-index LSB (NOT its Y-coord, but it works) to look up patterns from "Monos", and stores them into double hi-res memory.

If you would like to look at an attempt to convert lo-res to regular color double hi-res, you can change the monochrome routine to perform in color. First you should move the STA Column80On in "c" up four lines so that it is before the first STA AN3Off instead of the last one. This turns on mode 2 instead of mode 1. You will also need to include two more lines in subroutine ".2" of "ConvertLoResToDoubleHiRes560", and add a small table. I would put the table just before "Monos". Because the sample colors drawn by "Plot" (being 4 pixels wide) do convert correctly, be sure that there is a lot of text on the screen when you run the modified code. Change as follows:

```
.2   TAX
     LDA Spin,X
     ROL       (.2 moved up from this line)
...rest of s/r...
Spin >hs 00.08.01.09.02.0A.03.0B
     >hs 04.0C.05.0D.06.0E.07.0F
```

I've put the F/B hi-res mode to use many times to run programs like Asteroid Field (an oldie, but a goodie) and MousePaint on my RGB monitor. These programs were intended to be viewed on a monochrome monitor and, without help, they display unintended colors on RGB. By setting up F/B hi-res to display white on black (grey on brown looks good too) and then booting, I save having to turn on the monochrome NEC and turning my head to the left while playing (or painting). Here's how:

```
     *C001:0 N C055:0 N C057:0 N 400:F0 N 401<400.4000M
      C005:0 N 4000<2000.4000M C004:0 N
      C054:0 N C000:0 N C05E:0 N 6^P

or   *1/400:f0 n 1/401<1/400.6000m c05e 6^p  (EDM ONLY)
```

These Monitor commands also write auxiliary text page 1. This is because there is a F/B 40-column text mode too!

```
                  *SAVE S.Lo.to.HiRes
26-               LBas                 .eq $26,$27
2A-               HBas                 .eq $2A,$2B
2E-               ctr                  .eq $2E,$2F!!!
30-               Color                .eq $30
C000-             Store80Off           .eq $C000
C001-             Store80On            .eq $C001
C00C-             Column80Off          .eq $C00C
C00D-             Column80On           .eq $C00D
C01F-             ReadColumn80         .eq $C01F
C054-             MainPage             .eq $C054
C055-             AuxPage              .eq $C055
C05E-             AN3Off               .eq $C05E
C05F-             AN3On                .eq $C05F
                  *--------------------------------
0800- 20 80 09    a      JSR Plot
0803- 48                 PHA           save 40/80 state
0804- 20 66 08    .1     JSR Pause2
0807- 8D 5E C0           STA AN3Off    (double (?) hi-res on)
080A- AD 57 C0           LDA $C057     hi-res (RGB F-B Hi-Res)
080D- 20 9A 08           JSR ConvertLoResToFBHiRes
0810- 20 75 08           JSR Pause
0813- D0 EF              BNE .1        branch always
                  *--------------------------------
0815- 20 80 09    b      JSR Plot
0818- 48                 PHA           save 40/80 state
0819- 20 66 08    .1     JSR Pause2
081C- 8D 0D C0           STA Column80On    RGB MODE "4"
081F- 8D 5E C0           STA AN3Off        (160 Res 16 Color
0822- 8D 5F C0           STA AN3On          Double Hi-Res)
0825- 8D 0C C0           STA Column80Off
0828- 8D 5E C0           STA AN3Off
082B- 8D 5F C0           STA AN3On
082E- 8D 0D C0           STA Column80On
0831- 8D 5E C0           STA AN3Off    (double hi-res on)
0834- AD 57 C0           LDA $C057     hi-res
0837- 20 CF 08           JSR ConvertLoResToDoubleHiRes160
083A- 20 75 08           JSR Pause
083D- D0 DA              BNE .1        branch always
                  *--------------------------------
```

```
083F- 20 80 09      c       JSR Plot
0842- 8C 7E C0              sty $c07e      //c...
0845- 48                    PHA            save 40/80 state
0846- 20 66 08      .1      JSR Pause2
0849- 8D 5E C0              STA AN3Off           RGB MODE 1
084C- 8D 5F C0              STA AN3On            (560 Res Monochrome
084F- 8D 5E C0              STA AN3Off            Double Hi-Res)
0852- 8D 5F C0              STA AN3On
0855- 8D 0D C0              STA Column80On
0858- 8D 5E C0              STA AN3Off     (double hi-res on)
085B- AD 57 C0              LDA $C057      hi-res
085E- 20 04 09              JSR ConvertLoResToDoubleHiRes560
0861- 20 75 08              JSR Pause
0864- D0 E0                 BNE .1         branch always
                    *--------------------------------
0866- AD 56 C0      Pause2  LDA $C056      lo-res
0869- 8D 0C C0              STA Column80Off
086C- 8D 5F C0              STA AN3On      (double hi-res off)
086F- A9 04                 LDA /$400
0871- 85 27                 STA LBas+1
0873- 85 2E                 STA ctr        page counter too
0875- EE 00 04      Pause   inc $400       lo-res active mark rqrd for "a"
0878- AD 00 C0              LDA $C000      wait for any key
087B- 10 F8                 BPL Pause      ...not yet
087D- 8D 10 C0              STA $C010      clear strobe
0880- C9 8D                 CMP #$8D       clear Z-flag
0882- D0 15                 BNE .2          if NOT <return>
0884- 68                    PLA            pop return address
0885- 68                    PLA
0886- 8D 5F C0              STA AN3On      (double hi-res off)
0889- AD 51 C0              LDA $C051      text
088C- 68                    PLA            recover 40/80 state
088D- 30 07                 BMI .1         --->was 80-col
088F- 8D 0C C0              STA Column80Off      rqrd for "b" and "c"
0892- 8D 00 C0              STA Store80off too
0895- 60                    RTS            w/80 store off for 40-cols
0896- 8D 0D C0      .1      STA Column80On /
0899- 60            .2      RTS            w/80 store and 80-cols on
```

```
                   1790 ConvertLoResToFBHiRes
089A- A9 20        1800           LDA /$2000    hi-res page 1
089C- 85 2B        1810           STA HBas+1
089E- 98           1820      tya  LDA #$00      even GR rows all 0 (background)
089F- 20 C2 08     1830           JSR .3
08A2- A9 7F        1840           LDA #$7F      odd GR rows all 1 (foreground)
08A4- 20 C2 08     1850           JSR .3
08A7- 20 C2 09     1860 .1        JSR RBSC      picks lo-res too
08AA- A2 08        1870           LDX #8        hi-res lines/lo-res row pairs ctr
08AC- 2C 55 C0     1880           BIT AuxPage   fill aux mem w/color nybbles
08AF- 20 CB 09     1890 .2        JSR NxtH      store hi-res F-B pair/row pairs
08B2- CA           1900           DEX
08B3- D0 FA        1910           BNE .2
08B5- 2C 54 C0     1920           BIT MainPage
08B8- C8           1930           INY
08B9- D0 EC        1940           BNE .1
08BB- E6 27        1950           INC LBas+1
08BD- C6 2E        1960           DEC ctr
08BF- D0 E6        1970           BNE .1        loop for whole screen
08C1- 60           1980           RTS
08C2- A2 10        1990 .3        LDX #$10      page counter
08C4- 91 2A        2000 .4        STA (HBas),Y  write main pixels
08C6- C8           2010           INY
08C7- D0 FB        2020           BNE .4
08C9- E6 2B        2030           INC HBas+1
08CB- CA           2040           DEX
08CC- D0 F6        2050           BNE .4
08CE- 60           2060           RTS
                   2070 ConvertLoResToDoubleHiRes160
08CF- 20 C2 09     2080 .1        JSR RBSC      picks lo-res too
08D2- 29 0F        2090           AND #$0F      isolate even row nybble
08D4- 85 30        2100           STA Color
08D6- 4A           2110           LSR           line-up to left of msb
08D7- 20 ED 08     2120           JSR .2
08DA- B1 26        2130           LDA (LBas),Y  pick again
08DC- 29 F0        2140           AND #$F0      isolate odd row nybble
08DE- 85 30        2150           STA Color
08E0- 20 ED 08     2160           JSR .2
08E3- C8           2170           INY
08E4- D0 E9        2180           BNE .1
08E6- E6 27        2190           INC LBas+1
08E8- C6 2E        2200           DEC ctr
08EA- D0 E3        2210           BNE .1        loop for whole screen
08EC- 60           2220           RTS
08ED- 6A           2230 .2        ROR
08EE- 6A           2240           ROR
08EF- 6A           2250           ROR
08F0- 6A           2260           ROR
08F1- 05 30        2270           ORA Color     dup in other nybble
08F3- A2 04        2280           LDX #4        hi-res lines/lo-res row ctr
08F5- 2C 55 C0     2290 .3        BIT AuxPage
08F8- 91 2A        2300           STA (HBas),Y  write aux $2XXX (or $3XXX)
08FA- 2C 54 C0     2310           BIT MainPage
08FD- 20 CB 09     2320           JSR NxtH      write main $2XXX (or $3XXX) too
0900- CA           2330           DEX
0901- D0 F2        2340           BNE .3
0903- 60           2350           RTS
                   2360 * and for completeness:
                   2370 ConvertLoResToDoubleHiRes560
0904- 98           2380 .1        TYA
0905- 6A           2390           ROR           "column" lsb to carry
0906- 08           2400           PHP           save for odd rows
0907- 20 C2 09     2410           JSR RBSC      picks lo-res too
090A- 29 0F        2420           AND #$0F      isolate even row nybble
090C- 20 23 09     2430           JSR .2
090F- B1 26        2440           LDA (LBas),Y  pick again
0911- 4A           2450           LSR           isolate odd row nybble in LSN
0912- 4A           2460           LSR
0913- 4A           2470           LSR
0914- 4A           2480           LSR
0915- 28           2490           PLP           recover "column" lsb
0916- 20 23 09     2500           JSR .2
0919- C8           2510           INY
091A- D0 E8        2520           BNE .1
091C- E6 27        2530           INC LBas+1
091E- C6 2E        2540           DEC ctr
0920- D0 E2        2550           BNE .1        loop for whole screen
0922- 60           2560           RTS
```

```
0923- 2A           .2      ROL            double, merging "column" bit
0924- 0A                   ASL            double again for 4 bytes each
0925- AA                   TAX             and shove it in an index reg
0926- A9 04                LDA #4         hi-res lines/lo-res row ctr
0928- 85 2F                STA ctr+1      another counter loc...
092A- 2C 55 C0     .3      BIT AuxPage
092D- BD 40 09             LDA Monos,X
0930- 91 2A                STA (HBas),Y write aux $2XXX (or $3XXX)
0932- 2C 54 C0             BIT MainPage
0935- BD 41 09             LDA Monos+1,X
0938- 20 CB 09             JSR NxtH       write main $2XXX (or $3XXX) too
093B- C6 2F                DEC ctr+1      should I have used Color?
093D- D0 EB                BNE .3
093F- 60                   RTS
                           .ma hs         ".gen (off)" macro
                           .hs ]1
                           .em
0940-              Monos   >hs 00.00.00.00 0--black
0944-                      >hs 11.22.44.08 1--magenta
0948-                      >hs 22.44.08.11 2--dark blue
094C-                      >hs 33.66.4C.19 3--purple
0950-                      >hs 44.08.11.22 4--dark green
0954-                      >hs 55.2A.55.2A 5--grey (1)
0958-                      >hs 66.4C.19.33 6--meduim blue
095C-                      >hs 77.6E.5D.3B 7--light blue
0960-                      >hs 08.11.22.44 8--brown
0964-                      >hs 19.33.66.4C 9--orange
0968-                      >hs 2A.55.2A.55 A--grey 2 (tan!)
096C-                      >hs 3B.77.6E.5D B--pink
0970-                      >hs 4C.19.33.66 C--light green
0974-                      >hs 5D.3B.77.6E D--yellow
0978-                      >hs 6E.5D.3B.77 E--aquamarine
097C-                      >hs 7F.7F.7F.7F F--white
0980- A9 CC        Plot    LDA #$CC       start @ top-right
0982- A0 10                LDY #16
0984- 18           .1      CLC
0985- A2 03                LDX #3
0987- 85 30                STA Color      00, 44, 88, CC
0989- 88           .2      DEY
098A- 99 00 04             STA $400,Y     GR rows 0-3
098D- 99 80 04             STA $480,Y
0990- 69 11                ADC #$11       11, 55, 99, DD
0992- 99 00 05             STA $500,Y     GR rows 4-7
0995- 99 80 05             STA $580,Y
0998- 69 11                ADC #$11       22, 66, AA, EE
099A- 99 00 06             STA $600,Y     GR rows 8-11
099D- 99 80 06             STA $680,Y
09A0- 69 11                ADC #$11       33, 77, BB, FF
09A2- 99 00 07             STA $700,Y     GR rows 12-15
09A5- 99 80 07             STA $780,Y
09A8- A5 30                LDA Color      00, 44, 88, CC
09AA- CA                   DEX
09AB- 10 DC                BPL .2
09AD- 69 BC                ADC #-$44    end, 00, 44, 88
09AF- B0 D3                BCS .1         ...more
09B1- 84 2A        com...  STY HBas       Y-reg zero!
09B3- 84 26                STY LBas
09B5- AD 52 C0             LDA $C052      solid (40 x 48 pixels)
09B8- 8D 01 C0             STA Store80On     page 1 (main) assumed
09BB- AD 50 C0             LDA $C050      graphics
09BE- AD 1F C0             LDA ReadColumn80
09C1- 60                   RTS
09C2- A5 27        RBSC    LDA LBas+1  Bob's trick
09C4- 49 24                EOR /$2000^$400
09C6- 85 2B                STA HBas+1
09C8- B1 26                LDA (LBas),Y pick lo-res even-odd row pair
09CA- 60                   RTS
09CB- 91 2A        NxtH    STA (HBas),Y store hi-res...
09CD- E6 2B                INC HBas+1    new-old trick
09CF- E6 2B                INC HBas+1
09D1- E6 2B                INC HBas+1
09D3- E6 2B                INC HBas+1
09D5- 60                   RTS
                   *--------------------------------
```

Another Way Around the BRUN Problem.......Anonymous Caller

This morning I turned on my phone answering machine to play back the overnight messages. An anonymous caller calmly said, "This is an anonymous tip about the the BRUN problem under DOS. You can just BLOAD the program. Then CALL 41876, and the program will run properly and return nicely. You don't have to know the exact load address."

I checked it out, and it looks like our caller is correct. The BRUN processor inside DOS looks like this:

```
A38E- JSR BLOAD
A391- JSR IOHOOK
A394- JMP ($AA72)
```

By doing the complete BLOAD first, we have solved the problem of the trailing carriage return being printed after running our BRUNnable code. By the CALL 48176 (48176=$A394), we solve the problem of not knowing where the code loaded. In effect we have created a new BRUN command, which leaves out the call to IOHOOK and completes its own echoing before executing our code.

Thank you, Mr. Anonymous, whoever you are!

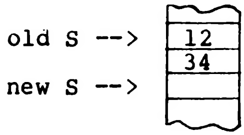
PEA, PEI, PER Instructions in 65802/16....Bob Sander-Cederlof

These are three new instructions for pushing data on the stack which need more explanation than you find in the data sheet of these new chips. Furthermore, some mis-information has gotten out about them.

All three push two bytes of data on the stack. Furthermore, it makes no difference whether you are in Emulation or Native mode, except for a minor problem which I will mention later.

PEA (Push Effective Address) pushes the two bytes which follow the PEA opcode onto the stack. It is, in effect, a "push 16-bit-immediate-value" instruction. Here is an example, with the equivalent in old-fashioned 6502 code also shown:

```
0800- F4 34 12   PEA $1234

0800- A9 12      LDA /$1234
0802- 48         PHA
0803- A9 34      LDA #$1234
0805- 48         PHA
```



Notice that PEA takes 3 bytes and 5 cycles, while the equivalent 6502 code takes 6 bytes and 10 cycles.

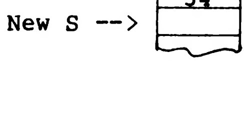
If you were in native mode with the m-status-bit cleared (that is, with A-register in 16-bit mode), and forgot about the PEA opcode, you might do this:

```
0800- A9 34 12   LDA ##$1234
0803- 48
```

This approach takes 4 bytes and 7 cycles, and has the advantage or dis-advantage of leaving the data also in the A-register.

PEI (Push Effective Indirect) pushes two bytes from the direct page onto the stack. You might think of it as "push (dir)". The byte at dir+1 is pushed first, and then the byte at dir. Here is an example, with equivalent 6502 code:

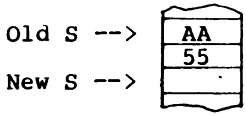
```
Assume D=$0000, $0055 contains $34
                $0056 contains $12

0800- D4 55      PEI $55

0800- A5 56      LDA $56
0802- 48         PHA
0803- A5 55      LDA $55
0805- 48         PHA
```



PEI takes 2 bytes and 6 cycles, and the equivalent 6502 code takes 6 bytes and 12 cycles. In Native 16-bit mode, you could use LDA $55 to pick up both bytes, and a single PHA to push both of them. This would take 3 bytes and 8 cycles.

PER (Push Effective Relative) is very similar to the PEA instruction. By analogy, PER is to PEA as BRL is to JMP. The processor adds the value in the two bytes following the opcode to the contents of the PC-register, and pushes the result (high-byte first) onto the stack. Before the addition takes place, the PC-register will have already been advanced to the first byte of the next instruction. Here is an example:

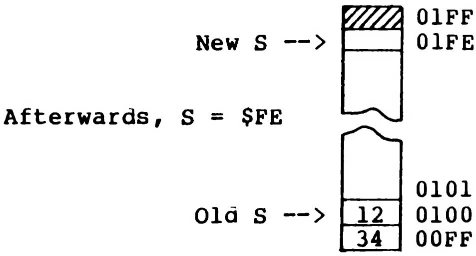
```
0800- 62 52 A2  PER $AA55
0803-
```



The value of PC after reading the PER instruction will be $0803. Adding $0803 to $A252 gives $AA55.

There is an idiosyncrasy (bug?) in the 65802 and I presume in the 65816 in the emulation mode, having to do with instructions which push two bytes on the stack. The three instructions discussed here (PEA, PEI, and PER), as well as the PHD instruction, all exhibit this problem. If the stack pointer is at $00 (remember, in emulation mode the stack pointer is only 8-bits wide and always points into page 1), you would expect the two bytes to be stored at $0100 and $01FF. However, the processor stores the "high" byte at $0100 as it should, and then stores the "low" byte at $00FF instead of $01FF. The following code will prove it to you, if you have the chip in your computer:

```
LDX #$00
STX $FF
STX $1FF
TXS
PEA $1234
LDA $FF
JSR $FDDA
LDA $100
JSR $FDDA
LDA $1FF
JSR $FDDA
JMP $3D0
```



The program above will print out "341200". This means that the PEA instruction pushed the value $34 into $00FF, rather than $01FF. If you plan to use these instructions in emulation mode, be sure you are aware of this un-expected case.

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.